Australian Government
Department of Defence
Defence Science and
Technology Organisation

Simulation of Statistical Distributions using the Memoryless Nonlinear Transform

Graham V. Weinberg and Lachlan Gunn

Electronic Warfare and Radar Division

Defence Science and Technology Organisation

DSTO–TR–2517

ABSTRACT

In support of Task 07/040 (support to AIR 7000), the generation of correlated sea clutter returns using the Memoryless Nonlinear Transform is investigated. The generation of such clutter is critical to the performance analysis of radar detection schemes under realistic clutter scenarios. This feature must be incorporated into clutter models being built at DSTO to test radar detector performance. Examples of the transform's application is given for a number of target distributions of interest.

Published by

DSTO Defence Science and Technology Organisation

PO Box 1500

Edinburgh, South Australia 5111, Australia

Telephone: (08) 7389 5555

Facsimile: (08) 7389 6567

AR No. AR-014-934

March 2011

APPROVED FOR PUBLIC RELEASE

Simulation of Statistical Distributions using the Memoryless Nonlinear Transform

Executive Summary

This work supports the modelling and simulation requirements for Task 07/040 (support to AIR 7000). Specifically, in order to test and evaluate the performance of radar detection schemes being considered at DSTO, it is necessary to evaluate their performance in a correlated clutter setting. Models of interest for sea clutter correspond to those being investigated for high grazing angle detection, specifically the K- and KK-Distribution models. Realistic sea clutter returns exhibit correlations in time, and it is the purpose here to show how to generate correlated K and KK-distributed clutter by simulation.

The Memoryless Nonlinear Transform (MNLT) provides a general framework which permits the generation of correlated sea clutter returns with desired marginal distributions. Although distributions that fit into the class of spherically invariant random processes (SIRV) can be simulated easily without the MNLT, some distributions of interest to DSTO may not fit within the class of SIRVs. Hence it is important to have an accurate description of the MNLT.

It is also shown how shortcomings of the MNLT can be overcome. Specifically, it has been reported that the MNLT is problematic because it does not guarantee an exact form of desired output autocovaiance. Essentially, the MNLT converts a correlated Gaussian process into a new process with desired marginal distributions. It is also correlated, but not through the same autocovariance function as the Gaussian process. It is in fact related to the Gaussian's autocovariance by a non-linear mapping. In a practical situation we would like to specify a desired process with given autocovariance function. It will be shown how this can be achieved, using an inversion method. This leads to a novel solution to the generation of correlated returns, with desired marginal distributions and prescribed autocovariance function.

The report outlines both the theoretical aspects of this transform, and how it is implemented in practice. A simulator has been developed, which is fully described and documented in this report. A number of simulations are included to show how the code can

be utilised. The simulator, written in Matlab, is in a form that will enable incorporation into a number of radar models under development at DSTO.

Authors

Graham V. Weinberg
EWRD

Graham V. Weinberg, BSc(Hons) PhD *Melb.,* MSc(Def SIP) *Adel.* is an S&T5 Mathematician working in the Microwave Radar Branch of EWRD. His research specialisation is predominantly in the area of high grazing angle detection of maritime targets from airborne surveillance platforms.

Lachlan Gunn
SVS-EWRD

Lachlan Gunn is a third year Bachelor of Engineering (Electronic and Electrical) and Bachelor of Mathematics and Computer Science Degree student at The University of Adelaide, working on a Summer Vacation Studentship.

Contents

Appendices

Figures

Tables

1 Introduction

The simulation of sea clutter returns is an integral part of the modelling, analysis and validation of radar systems. In particular, it is important to have the capability to generate synthetic radar clutter that not only posesses the desired marginal distributional characteristics, but also exhibits what can be termed short range correlations in time. By the latter we refer to clutter returns whose covariances decrease with time, such as through, for example, a damped sinusoidal function.

The purpose of this report is to show how the Memoryless Nonlinear Transform (MNLT) [1] can be used to simulate amplitude distributions with correlation in time. The MNLT is a nonlinear mapping from a specified probability distribution to a target, or desired distribution. Its formulation means we have the flexibility to start with *any* distribution and convert it into a desired distribution. As such, we can begin with a correlated Gaussian stochastic process, and produce a correlated process having the distributional characteristics of interest.

The literature contains a number of cases where the MNLT has been used to generate correlated returns. One of the earliest references is [2], who uses the MNLT to correlate Log-Normal clutter. Additionally, [3] also examines the Log-Normal case, while [4] and [5] extend the MNLT to the Weibull case. The extension of the MNLT to the simulation of correlated K-Distributed clutter first appeared in [6], followed by the derivation of the relationship between correlation functions in [1].

In the DSTO report [7] it is argued that the MNLT does not provide an entirely sufficient solution to the simulation of correlated clutter[1]. Part of the criticism of the method is that it does not indicate how the correlation characteristics of the target distribution can be controlled. This is because we begin with a correlated Gaussian process, and produce an output process with a correlation function that has been transformed from the Gaussian correlation function via the MNLT. However, it has been shown in [1] that the correlation function of the target distribution can be expanded in terms of the Gaussian

[1]It is worth observing that [1] appeared after the DSTO report [7] and so the author of the latter was unaware of the former.

correlation function. Specifically, the expansion is in terms of Hermite polynomials from mathematical physics. This relationship, as described in [1], can in principle be used to determine which Gaussian correlation should be used to produce the desired correlations in the target distribution sample. The issue with the relationship is that it expresses the output autocovariance as a function of the Gaussian process autocovariance. What is required is the inverse mapping, which will then specify which Gaussian process one should start with. The novelty in the work presented here is that a general methodology, supported by Matlab code, is developed to achieve this aim. It involves an approximate inversion of the correlation mapping from [1], which then enables the determination of an appropriate Gaussian input process.

In addition to the preceived issues with determining which Gaussian process to use as input, it is also argued in [7] that the theory of spatially invariant random processes (SIRP) provides a better solution to the problem under consideration. While this is true for the class of SIRPs, the problem is that there are distributions of interest to DSTO that *may not fit into this class*. The prime example is the KK-Distribution [20], which is a model for high grazing angle clutter. At the time of writing it was not clear whether the KK-Distribution fits within this class. Hence it is important to examine whether the MNLT is useful for simulating a correlated sequence with KK-Distributed marginals.

This report analyses the problem of generating correlated clutter returns from a mathematical point of view, but also develops a practical mechanism for generating it. Included is a correlated clutter simulator in Appendix B.

To clarify what is the purpose of this work, we formulate the desired outcome in the following manner. We would like to generate a random sample of observations $x_1, x_2, \ldots, x_n$ from a prescribed continuous random variable X, with density function $f_X(t)$, cumulative distribution function $F_X(t)$ and covariance $\mathrm{Cov}(X_i, X_j) = \mathbb{E}(X_i X_j) - \mathbb{E}(X_i)\mathbb{E}(X_j)$, where the random variables corresponding to the realisations x_i and x_j are X_i and X_j respectively. These realisations represent amplitude or intensity measurements. We refer to this stochastic process as the target or output process, since it is the result of the MNLT. The Gaussian stochastic process used to generate the target process is called the input process. We will restrict attention to stationary processes throughout, so that the mean of each

realisation is constant. We will employ the notation and terminology used in [1], referring to $r(k) = \mathbb{E}(X_0 X_k)$ as the correlation function of the process. We furthermore define the covariance coefficient of the process as $(r(k) - \mathbb{E}(X_0)\mathbb{E}(X_k))/\mathrm{Var}(X)$.

Such a series of observations can be used as range samples from a single-pulse return that are input to a detection scheme, such as a Constant False Alarm Rate (CFAR) detector, or as time samples from multiple pulses in a single range cell. We do not examine long-term correlations in this work.

This report is structured as follows. Section 2 introduces the MNLT, and Section 3 examines the relationship between correlation functions, and shows how they can be related analytically in a special case. Section 4 discusses our implementation of the algorithms in question, while Section 5 discusses some examples of its output. Appendix A describes the use of the simulator from the point of view of the user, while Appendix B contains the source code for the simulator. Finally, Appendix C contains numerically determined values related to the function described in Section 3.

Due to the fact that this work is highly probabilistic in nature, it is worthwhile listing some useful references on statistics, probability and simulation. Suitable references on probability and stochastic processes include [8] , [9] and [10], while [11] is a useful reference on simulation. Statistical tests and methods are described in [12]. Standard radar texts all describe sea clutter and their characteristics, for example [13] and [14] both describe sea clutter in depth.

2 Simulation via The Memoryless Nonlinear Transform

The MNLT is essentially an extension of a fundamental method with which a single random variable return can be generated. We thus begin with the generation of independent returns, and show how this can be modified to produce dependency between the marginal distributions in the process being simulated. Throughout, we are focusing on continuous random variables as specified in Section 1. We also employ the notation $X \stackrel{d}{=} Y$ to mean that the random variables X and Y, defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$, share the same distribution function. This means that $\forall A \in \mathcal{F}$, $F_X(A) = \mathbb{P}(X \in A) = F_Y(A) = \mathbb{P}(Y \in A)$.

The following Lemma is the first key simulation result:

Lemma 2.1 *Suppose X is a continuous random variable with cumulative distribution function F_X, and let $R \stackrel{d}{=} R(0,1)$ be a uniformly distributed random variable on the unit interval. Then the random variable $F_X^{-1}(R) \stackrel{d}{=} X$.*

To see this, let $Y = F_X^{-1}(R)$ and observe that the cumulative distribution function of Y is

$$F_Y(y) = \mathbb{P}(F_X^{-1}(R) \leq y) = \mathbb{P}(R \leq F_X(y)) = F_X(y), \tag{1}$$

using the cumulative distribution function of a uniform random variable. This implies $Y \stackrel{d}{=} X$, as required.

Lemma 2.1 states that to generate a realisation from a distribution X we need only generate a uniform random number between 0 and 1 and evaluate the inverse of the cumulative distribution function at this point. Since we are dealing with continuous distributions, the cumulative distribution function will be monotonically increasing and continuous, and so the inverse will always exist.

To illustrate how (1) can be used to simulate a random variable, consider the case where X is an exponential distribution with density $f_X(x) = e^{-x}$, for $x \geq 0$. Then it is not difficult to show that its cumulative distribution function $F_X(x) = 1 - e^{-x}$ and $F_X^{-1}(x) =$

$-\log(1-x)$. Hence Lemma 2.1 implies that if $r \in [0,1]$ is a random number, then $-\log(1-r)$ is a realisation of X. Generating a large sequence of such numbers, and plotting a histogram, will show that they have the same statistical characteristics as realisations from an exponential distribution. It is worth noting that if r is a random number between 0 and 1, so is $1-r$ and so for simulation purposes we can focus on generating $-\log(r)$ in the above. This result is clarified in the following Lemma:

Lemma 2.2 *If* $R \stackrel{d}{=} R(0,1)$*, then* $1-R \stackrel{d}{=} R(0,1)$*.*

Lemma (2.2) is proven easily by constructing the distribution function of $1-R$. Another useful result is presented in Lemma 2.3, which is essentially equivalent to Lemma 2.1:

Lemma 2.3 *If* X *is a continuous random variable with cumulative distribution function* F_X *and* $R \stackrel{d}{=} R(0,1)$*, then* $F_X(X) \stackrel{d}{=} R$*. That is, the random variable resulting by substituting* X *into its distribution function is uniformly distributed.*

This Lemma is proven by showing that the distribution function of $F_X(X)$ matches that of R.

Lemma 2.1 can be used to generate *independent* samples. Recalling the problem specification from Section 1, we want to generate a correlated sequence, whose point to point distributions match a prescribed distribution. It is clear the only way to proceed would be to generate a correlated sequence of uniform random numbers, and then apply Lemma 2.1 to produce the sequence with target marginal distribution. This sequence will be thus correlated. However, we can use Lemma 2.3 together with Lemma 2.1 to do this more generally. The key to this is the following version of Lemma 2.1:

Lemma 2.4 *Suppose* X_1 *and* X_2 *are two random variables, with distribution functions* F_{X_1} *and* F_{X_2} *respectively. Then the random variable* $F_{X_2}^{-1}(F_{X_1}(X_1)) \stackrel{d}{=} X_2$*.*

This Lemma follows by applying Lemma 2.3 to Lemma 2.1.

Lemma 2.2 is the key to generating correlated realisations of a random process. Define a function $\eta(x) = F_{X_2}^{-1}(F_{X_1}(x))$. Then Lemma 2.2 implies that if x is a realisation of

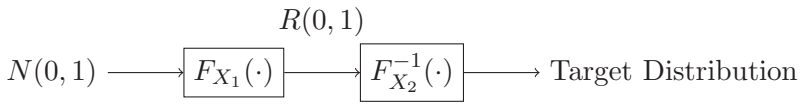

Figure 1: *The Memoryless Nonlinear Transform converts a Gaussian variable into one of the target distribution.*

random variable X_1, then $\eta(x)$ is a realisation of random variable X_2. Suppose we have a wide sense stationary stochastic process $\zeta(t)$ evolving over time t, with correlation function $r_\zeta(\tau) = \mathbb{E}(\zeta(0)\zeta(\tau))$, and each $\zeta(t) \stackrel{d}{=} X_1$. Then we can transform this process to produce a new stochastic process $\theta(t) = \eta(\zeta(t))$, which will be distributed according to X_2 pointwise ($\theta(t) \stackrel{d}{=} X_2$, $\forall t \geq 0$). However, this new process will have a correlation function $r_\theta(\tau) = \kappa(r_\zeta(\tau))$, where $\kappa(\cdot)$ is a nonlinear function.

The utility of this and Lemma 2.2 is that we can generate *any* random variable X_2 from any X_1. Hence, we choose X_1 to be a Gaussian random variable, and generate a correlated Gaussian sequence. This is a relatively straightforward exercise, see for example [15]. This is then transformed into a sequence of random variables with the desired statistical distribution using the function η. Figure 1 illustrates the MNLT process graphically.

The resulting sequence will consist of dependent realisations. Due to the nonlinear nature of η, the output sequence will not necessarily have the same correlation properties as the Gaussian input process. This is examined in the next Section.

3 Correlation Function Mapping

This Section is concerned with the autocorrelation properties of the MNLT, and in particular, investigates the relationship between the Gaussian input and target distribution output autocorrelation. As remarked previously, [1] derives an expansion of the output autocorrelation in terms of the input autocorrelation. What is desired is an inverse mapping, so that we know which Gaussian process to begin with, in order to generate a desired stochastic process with specified marginal distributions and autocorrelation function. While this is not always possible to do analytically, it can be done through a series of functional approximations. We begin with the series representation of the output autocorrelation.

3.1 Correlation Function Expansion

In order to understand the relationship between the correlation functions related to the MNLT it is necessary to take a brief digression into some special functions from mathematical analysis. A useful reference on the latter is [16], although we merely require the properties of certain special functions for the work to follow. Hermite polynomials [16, 17] are a useful set of functions used in mathematical analysis, and in particular, in Hilbert space theory. They form a complete orthogonal sequence with respect to a Gaussian weight function, and arise in the theory of differential equations—in particular, as a special case of the Sturm-Liouville boundary value problem [17, 18]. The 0th order Hermite polynomial is $H_0(t) = 1$ and for $n \in \mathbb{N}$, the nth order Hermite polynomial is

$$H_n(t) = (-1)^n e^{t^2} \frac{d^n}{dt^n}(e^{-t^2}).$$

Hermite polynomials can also be generated with a second-order recurrence relation. Noting that $H_1(t) = 2t$, it can be shown that

$$H_{n+1}(t) = 2tH_n(t) - 2nH_{n-1}(t).$$

This enables us to sequentially generate Hermite polynomials. The generating function of the Hermite polynomials is

$$e^{(2xt-x^2)} = \sum_{n=0}^{\infty} \frac{1}{n!} H_n(t) x^n. \tag{2}$$

It is worth noting that the generator (2) essentially specifies a Gaussian density in terms of the Hermite polynomials.

The following Lemma is an interesting result that will be applied in the Lognormal example considered in the next Section:

Lemma 3.1 *Suppose* $X \stackrel{d}{=} N(\mu, 0.5)$ *is a Gaussian random variable with fixed mean* μ. *Then for each* $n \in \mathbb{N}$, $\mathbb{E}\left(H_n(X)\right) = (2\mu)^n$.

The proof follows directly from the generator function (2).

We now derive the relationship between Gaussian input and target output autocorrelation functions. It essentially follows from the generating function (2). Using equations (9) and (10) of [19], (with the choice of $j = 1$, $k = 1$, $z = \frac{r}{2}$, $x = \frac{x_1}{\sqrt{2}}$ and $y = \frac{y_1}{\sqrt{2}}$), it can be shown that

$$e^{-\frac{1}{2(1-r^2)}[x_1^2+y_1^2-2x_1y_1r]} = \sqrt{1-r^2}e^{-\frac{(x_1^2+y_1^2)}{2}} \sum_{n=0}^{\infty} H_n\left(\frac{x_1}{\sqrt{2}}\right) H_n\left(\frac{y_1}{\sqrt{2}}\right) \frac{r^n}{2^n n!} \tag{3}$$

As pointed out in [1], this equation gives an expansion of the generator of an Ornstein-Uhlenbeck process via eigenfunctions of its associated Fokker-Plank equation. When weighted by $2\pi\sqrt{1-r^2}$, the left hand side of (3) becomes the probability density function of a bivariate Gaussian process, with mean zero, variances one, and covariance r. Denote this density $f_{(Y_1,Y_2)}$. This means that the correlation function of the target process in the MNLT formulation can be written in terms of that of the input (Gaussian process) and Hermite polynomials. Let r_{input} be the Gaussian correlation function, so that $r_{input}(t_1, t_2) = \mathbb{E}(X_{t_1} X_{t_2})$, and let the output correlation be $r_{output}(t_1, t_2)$. In particular, using this and the definition of correlation function, it can be shown that for a Gaussian input process,

$$r_{output}(t_1, t_2) \quad = \quad \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \eta(y_1)\eta(y_2) f_{(Y_1,Y_2)}(y_1, y_2) dy_1 dy_2$$

n	a_n
0	$\mathbb{E}(X_2)$
1	$\sqrt{2}\mathbb{E}(X_1X_2)$
2	$2\left(\mathbb{E}(X_1^2X_2)-\mathbb{E}(X_2)\right)$
3	$2\sqrt{2}\left(\mathbb{E}(X_1^3X_2)-3\mathbb{E}(X_1X_2)\right)$

Table 1: *The first few coefficients expressed as expectations.*

$$= \frac{1}{\pi}\sum_{n=0}^{\infty}\frac{r_{input}(t_1,t_2)^n}{2^n n!}\left(\int_{-\infty}^{\infty}e^{-x^2}H_n(x)\eta(\sqrt{2}x)dx\right)^2$$

$$= \sum_{n=0}^{\infty}\frac{r_{input}(t_1,t_2)^n}{2^n n!}a_n^2, \tag{4}$$

with the coefficients a_n given by

$$a_n = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty}e^{-x^2}H_n(x)\eta(\sqrt{2}x)dx$$

$$= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}H_n\left(\frac{x}{\sqrt{2}}\right)\eta(x)dx$$

$$= \mathbb{E}\left[H_n\left(\frac{X_1}{\sqrt{2}}\right)\eta(X_1)\right], \tag{5}$$

the mean in (5) being with respect to the Gaussian marginal distribution $X_1 \stackrel{d}{=} N(0,1)$. Note that if we assume that the Gaussian input process is wide sense stationary, the output process will preserve this feature, in view of (4). Consequently we can write, in terms of discrete time $k \in \mathbb{N}$,

$$r_{output}(k) = \sum_{n=0}^{\infty}\frac{(r_{input}(k))^n}{2^n n!}a_n^2. \tag{6}$$

Using the definition of the Hermite polynomials, as well as their recurrence relation, it can be shown that the first few Hermite polynomials are $H_0(t)=1, H_1(t)=2t, H_2(t)=4t^2-2$ and $H_3(t)=8t^3-12t$. Applying these results to (5), we can show the first few coefficients are as given in Table 1.

The expression (6) expands the output correlation function in terms of the Gaussian input correlation function as a power series. It is reported in [1] that this series is rapidly

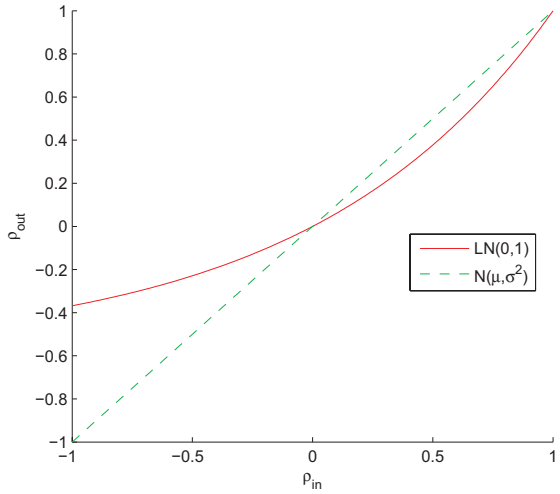

Figure 2: *Relationships between input and output autocovariance coefficients for the Log-Normal distribution.*

convergent for a number of target distributions of interest. This mapping between the two correlation functions shows how, in theory, we can control the output correlation through the Gaussian input correlation. In essence, we require an inversion of (6).

As can be seen in figure 2, this mapping is very much nonlinear, and thus inversion in the general case is clearly not a simple exercise. Despite this difficulty, as pointed out in [1] and also indicated in [6], we can establish some "rules of thumb" to determine which Gaussian process to use as input. This is addressed in the following section.

3.2 Methods of Inversion

Rules of thumb are now outlined for inversion of the autocorrelation relationship derived in the previous Section. Firstly, it has been reported in [1, 6] that for some clutter models of interest, there is an approximate invariance between input and output correlations. The Weibull distribution is a good example of this. It has been found in many situations, there is an almost linear association between the correlation functions. This means we can apply a truncation of (6) to produce the linear estimate

$$r_{output}(k) \approx (\mathbb{E}(X_2))^2 + (\mathbb{E}(X_1 X_2))^2 r_{input}(k), \tag{7}$$

and inversion is immediate, noting it may be necessary to scale the linearisation to produce a proper correlation function for the Gaussian (which means at $k = 0$ it must equal 1, the Gaussian's variance). A second approach is to use *ad hoc* adjustment of the input correlation to produce an output with local dependency as described by a sample correlation. This assumes we are not overly concerned about the specific form of the output correlation, but require a correlated sequence with the correct marginal distributions. A third approach is to use a higher order polynomial approximation to (6), such as a cubic or quartic approximation, and apply inversion numerically, scaling the appropriate solution to produce an approximate Gaussian correlation function.

There is a fourth approach to designing the appropriate Gaussian input distribution. The Lagrange Inversion Theorem can be applied to invert the implicit function defined through the series (6). Suppose we have an analytic function f, and that at a point a the derivative $f'(a) \neq 0$. Then, the Theorem states that the equation $f(w) = z$ can be solved to give $w = g(z)$, where g has a series expansion

$$g(z) = a + \sum_{m=1}^{\infty} \lim_{w \longrightarrow a} D^{m-1} \left(\frac{w-a}{f(w)-f(a)} \right)^m \frac{(z-f(a))^m}{m!}, \tag{8}$$

and D^m is the mth derivative with respect to w. This can be applied to (6) by selecting $a = 0$ and letting $f(w) = \sum_{n=0}^{\infty} b_n w^n$, where $b_n = \frac{a_n^2}{2^n n!}$, and noting that f is analytic and $f'(0) = \mathbb{E}^2(X_1 X_2) \neq 0$ in general. Here $f(a) = f(0) = b_0$. As an example, four terms were manually calculated, resulting in the approximate solution

$$\begin{aligned} r_{input}(k) \quad \approx \quad & \frac{1}{b_1}(r_{output}(k) - b_0) - \frac{b_2}{b_1^3}(r_{output}(k) - b_0)^2 \\ & - \left(\frac{b_3}{b_1^4} - \frac{2b_2^2}{b_1^5} \right) (r_{output}(k) - b_0)^3 - \left(\frac{b_4}{b_1^5} - \frac{5b_2 b_3}{b_1^6} + \frac{5b_2^3}{b_1^7} \right) (r_{output}(k) - b_0)^4. \end{aligned} \tag{9}$$

Given a desired distribution F_{X_2} with correlation function r_{output}, equation (9) gives an indication of which Gaussian distribution X_1 should be used to generate the desired correlated sample. Observe that $b_0 = \mathbb{E}^2(X_2)$ and so the expansion (9) is in terms of the process autocovariance function, assuming we have a stationary target distribution.

Of the four approaches outlined here, we will focus exclusively on the third, which involves an inversion of a partial series of the correlation function. It is expected to give more

accurate results than a linearisation, while the approach based upon Lagrange Inversion is basically equivalent, but requires more analytical work.

3.3 Lognormal Distributed Clutter: A Special Case

Before proceeding to the approximate solutions to the inversion problem outlined in the previous Section, we examine a case where the relationship between input and output correlation can be given in closed form. In [1] there are several such examples given, but these are not entirely convincing. The example presented here is perhaps better because it indicates how the series in (6) can be evaluated for a distribution of interest in radar. We remark at the outset that the following simplification of (6) can be derived using conditional probability without reference to the series expansion (6).

A positive random variable X is said to be Lognormally distributed if its logarithm is Normally distributed. We write $X \stackrel{d}{=} LN(\mu, \sigma^2)$ where $\log(X) \stackrel{d}{=} N(\mu, \sigma^2)$. This distribution has been investigated in the past as a model for radar clutter [2, 3].

Let $X_1 \stackrel{d}{=} N(0,1)$ and $X_2 \stackrel{d}{=} LN(\mu, \sigma^2)$. Then we can select $X_2 = e^{\mu+\sigma X_1}$, and consequently $\eta(z) = e^{\mu+\sigma z}$. Simulating correlated returns is quite easy here because of the straightforward relationship between the input and output distributions. We now evaluate the coefficients a_n; note that by using the form of η and applying a change of variables,

$$\begin{aligned} a_n &= e^{\mu} \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} e^{-\frac{w^2}{2}+\sigma w} \frac{1}{\sqrt{2\pi}} H_n\left(\frac{w}{\sqrt{2}}\right) dw \\ &= e^{\mu+\frac{\sigma^2}{2}} \int_{-\infty}^{\infty} H_n\left(\frac{w}{\sqrt{2}}\right) \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-\sigma)^2)} dw \\ &= e^{\mu+\frac{\sigma^2}{2}} \mathbb{E}\left[H_n\left(\frac{W}{\sqrt{2}}\right)\right], \end{aligned} \tag{10}$$

where $W \stackrel{d}{=} N(\sigma, 1)$. We now apply Lemma 3.1, simplifying (10) to

$$a_n = e^{\mu+\frac{\sigma^2}{2}} 2^{\frac{n}{2}} \sigma^n. \tag{11}$$

Substituting into (6), we see that

$$r_{output}(k) = \sum_{n=0}^{\infty} \frac{[\sigma^2 r_{input}(k)]^n}{n!} e^{2\mu+\sigma^2}$$

$$= \quad e^{\sigma^2 r_{input}(k)+2\mu+\sigma^2}, \tag{12}$$

where the Taylor series expansion for the exponential function has been used. By taking logarithms of (12) we can obtain the required input correlation. This result is a useful test mechanism for other approximate schemes.

3.4 Hilbert Spaces and Parseval's Identity

In the analysis to follow, we will be exclusively using truncation of the series (6) and applying an approximate inversion to it. It will thus be important to quantify the error incurred by truncation of the series. In order to achieve this, it is useful to review some Hilbert space theory, which provides some useful tools for this purpose. In particular, we will require Parseval's Identity as well as an understanding of the representation of a function in terms of basis elements.

Let X be a vector space over either the real or complex numbers, with scalar field F. Then an *inner product* [16] is a function

$$\langle \cdot, \cdot \rangle : X \times X \to F$$

such that all of the following hold, for $x, y, z \in X$ and $\alpha \in F$:

$$\langle x + y, z \rangle \quad = \quad \langle x, z \rangle + \langle y, z \rangle \tag{13}$$

$$\langle \alpha x, y \rangle \quad = \quad \alpha \langle x, y \rangle \tag{14}$$

$$\langle x, y \rangle \quad = \quad \overline{\langle y, x \rangle} \tag{15}$$

$$\langle x, x \rangle \quad \geq \quad 0 \tag{16}$$

$$0 = \langle x, x \rangle \quad \iff \quad x = 0. \tag{17}$$

That is, it is a function from a pair of vectors to a scalar that is linear in its first argument (13,14), conjugate-symmetric (15), and positive-definite (16,17). While (16) might look somewhat suspicious at first, since the inner product is a complex-valued function, the application of (15) will show that $\langle x, x \rangle$ must be real.

A space X with an associated inner product $\langle \cdot, \cdot \rangle$ is called an *inner product space*. If it is also complete, it is called a *Hilbert space*. This means every Cauchy sequence in the space converges [16], with respect to the norm induced by the inner product, to a point in the space. The associated norm is $\|x\|^2 = \langle x, x \rangle$, and is a mapping $\| \cdot \| : X \to \mathbb{R}$.

If two vectors have an inner product of zero, we call them *orthogonal*. If the elements of a set of vectors are orthogonal to each other element, we say that the set is *mutually orthogonal*. If each element has a norm of one, we describe the set as *orthonormal*. If this set is sufficiently large that we can write any vector as a linear combination of its elements, we call it an *orthonormal basis*. The representation in terms of any particular basis is unique.

One might then ask how to find the coefficients of each member of the basis that are used when forming a particular vector. When the basis $\{e_n\}$ is orthonormal, this is

$$x = \sum_n e_n \langle x, e_n \rangle. \tag{18}$$

Having found this representation, we may then introduce the Parseval Identity, which will play an important role in determining the error bound on our truncated correlation series. Since $\|x\|^2 = \langle x, x \rangle$, it follows by (18) that

$$\begin{aligned} \|x\|^2 &= \left\langle \sum_n e_n \langle x, e_n \rangle, x \right\rangle \\ &= \sum_n \langle e_n \langle x, e_n \rangle, x \rangle \\ &= \sum_n \langle x, e_n \rangle \langle e_n, x \rangle \\ &= \sum_n \langle x, e_n \rangle \overline{\langle x, e_n \rangle} \\ &= \sum_n |\langle x, e_n \rangle|^2 . \end{aligned} \tag{19}$$

In Euclidean space, this statement essentially states that even if we select a different coordinate system to represent our vectors, the square of the length of a vector will still be given by the sum of the squares of its components as long as the new basis is orthonormal.

The family of spaces that we require for our application is $L^2(-\infty, +\infty)$, and an appropriate choice of basis functions is the Hermite polynomials with respect to the Gaussian

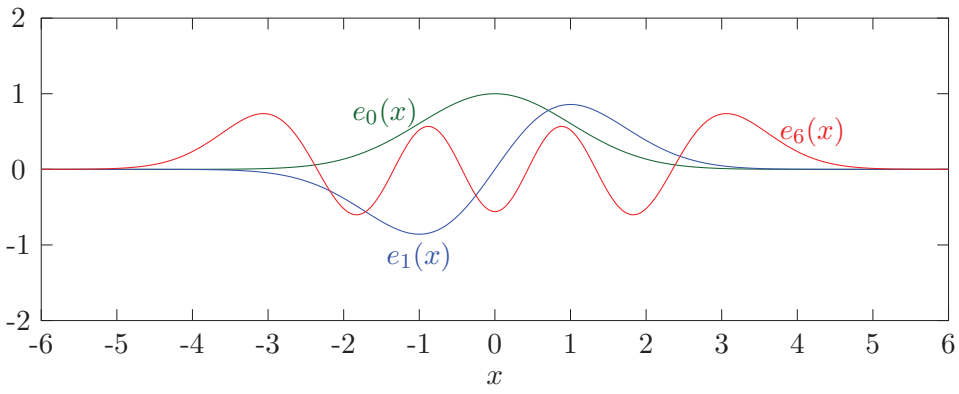

Figure 3: *The zeroth, first, and sixth Hermite functions.*

weight function (see figure 3), given by

$$e_n(x) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} e^{-\frac{1}{2}x^2} H_n(x), \tag{20}$$

where the inner product on this space is given by

$$\langle f, g \rangle = \int_{-\infty}^{+\infty} f(x)\overline{g(x)}\, dx \tag{21}$$

(see [16] for details of this, including justification that the sequence $\{e_n(x)\}$ forms an orthonormal basis).

We shall see that the effect of the MNLT is described most simply when it is considered in terms of these functions.

Before doing so, however, we return once more to Parseval's theorem, and its interpretation in the context of functions in $L^2(I)$, for a general interval I. The norm of a function f is

$$\begin{aligned} \|f\|^2 &= \int_I f(t)\overline{f(t)}\, dt \\ &= \int_I |f(t)|^2\, dt. \end{aligned} \tag{22}$$

The square of the norm of a function of $L^2(I)$ is therefore the total energy of the function.

Expanding (19) using (21, 22), we see that (where $\{e_n\}$ is an arbitrary orthonormal basis)

$$\int_I |f(t)|^2\, dt = \sum_n |\langle f, e_n \rangle|^2 . \tag{23}$$

We can write the coefficients for each element of the basis as a discrete sequence

$$c_n = \{\ldots, \langle f, e_{-1}\rangle, \underline{\langle f, e_0\rangle}, \langle f, e_1\rangle, \ldots\},$$

and therefore Parseval's theorem is the statement that the energy in this discrete signal is equal to the energy in our original function.

3.5 Inner Product Representation

The purpose here is to show how the correlation series (6) can be expressed in terms of the inner product and associated basis functions of the associated Hilbert space $L^2(-\infty, +\infty)$. This allows the application of Parseval's Identity to the series, and perhaps more importantly grants an understanding of the distribution of output autocorrelation amongst the infinitude of terms.

Note that the coefficients a_n of (5) can be written

$$\begin{aligned} a_n &= \frac{1}{\sqrt{\pi}} \int_{-\infty}^{+\infty} e^{-x^2} H_n(x) \eta(x\sqrt{2})\, dx \\ &= \frac{1}{\sqrt{\pi}} \int_{-\infty}^{+\infty} \left(e^{-\frac{1}{2}x^2} \eta\left(x\sqrt{2}\right)\right) \overline{\left(e^{-\frac{1}{2}x^2} H_n(x)\right)}\, dx \\ &= \frac{1}{\sqrt{\pi}} \left\langle e^{-\frac{1}{2}x^2} \eta\left(x\sqrt{2}\right), e^{-\frac{1}{2}x^2} H_n(x) \right\rangle. \end{aligned} \tag{24}$$

The Hermite functions are given by

$$e_n(x) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} e^{-\frac{1}{2}x^2} H_n(x),$$

and we define

$$\psi(x) = \frac{1}{\sqrt[4]{\pi}} e^{-\frac{1}{2}x^2} \eta\left(x\sqrt{2}\right),$$

allowing us to then simplify (24)

$$\begin{aligned} a_n &= \frac{1}{\sqrt{\pi}} \left\langle \psi \sqrt[4]{\pi}, e_n \sqrt{2^n n! \sqrt{\pi}} \right\rangle \\ &= \sqrt{2^n n!} \langle \psi, e_n \rangle \end{aligned} \tag{25}$$

and thus write (6) as

$$r_{out}(k) = \sum_{n=0}^{\infty} r_{input}(k)^n \langle \psi, e_n \rangle^2.$$

Knowing that both ψ and e_n are real, this is equivalent to

$$r_{out}(k) = \sum_{n=0}^{\infty} r_{input}(k)^n \left|\langle \psi, e_n \rangle\right|^2, \tag{26}$$

thus putting the series into a form amenable to application of Parseval's Identity.

As the input $X_1 \overset{d}{=} N(0,1)$, we have that $r_{input}(0) = \mathbb{E}\left(X_1^2\right) = 1$. Similarly, by the definition of autocorrelation, $r_{out}(0) = \mathbb{E}\left(X_2^2\right)$. Thus, we can write a special case of (26):

$$\mathbb{E}\left(X_2^2\right) = \sum_{n=0}^{\infty} \left|\langle \psi, e_n \rangle\right|^2. \tag{27}$$

Thus it follows from Parseval's Identity that

$$\|\psi\|^2 = \mathbb{E}\left(X_2^2\right), \tag{28}$$

as the Hermite functions form a complete orthonormal basis for $L^2(-\infty, +\infty)$. But, $\|\psi\|^2$ is the energy of the signal ψ, and therefore it has total energy equal to the instantaneous power of the target process.

One might perhaps choose to interpret the output autocorrelation as the energy in a filtered version of ψ. In essence, the multiplication by x^n corresponds to an attenuation by a factor of x^{-n} of each term n. When we attempt to bound the remainder, we do so by considering the power in ψ that the truncated series has failed to include.

3.6 Bounding the Truncation Error

The power series described in (26) can be truncated and used to determine the autocorrelation characteristics of the transformed process. We then desire to know the error that such an approximation introduces. Let $g_k(x)$ be the truncation of the series to order k; that is, let

$$g_k(x) = \sum_{n=0}^{k} x^n \left|\langle \psi, e_n \rangle\right|^2, \tag{29}$$

and the corresponding remainder function be

$$r_k(x) = \sum_{n=k+1}^{\infty} x^n \left|\langle \psi, e_n \rangle\right|^2. \tag{30}$$

It should be noted that e_n being orthonormal implies that setting $\psi = e_p$ will result in $\langle \psi, e_n \rangle = \delta_{np}$, and thus we may produce a relationship between input and output autocorrelation described by an arbitrary polynomial by selecting an appropriate linear combination of these functions. Adding to ψ the function ce_k will therefore produce a term with coefficient c at $n = k$, demonstrating that one cannot bound the remainder without some knowledge of the target distribution.

For an input autocorrelation of magnitude at most one, as in the case of a standard normal distribution, the worst-case error occurs when all the remaining energy of ψ is in the next (and least attenuated) term. Irrespective of the distribution of energy in ψ, for $n > k$ we have that $|x|^n \leq |x|^{k+1}$, and therefore can bound the remainder series:

$$\begin{aligned}
|r_k(x)| &= \left| \sum_{n=k+1}^{\infty} x^n \left| \langle \psi, e_n \rangle \right|^2 \right| \\
&\leq \sum_{n=k+1}^{\infty} |x^n| \left| \langle \psi, e_n \rangle \right|^2 \\
&\leq \sum_{n=k+1}^{\infty} \left| x^{k+1} \right| \left| \langle \psi, e_n \rangle \right|^2 \\
&= |x|^{k+1} \sum_{n=k+1}^{\infty} \left| \langle \psi, e_n \rangle \right|^2 \\
&= |x|^{k+1} r_k(1). \qquad (31)
\end{aligned}$$

By (27), we have that

$$g_k(1) + r_k(1) = \mathbb{E}\left(X_2^2\right),$$

and thus that

$$|r_k(x)| \leq |x|^{k+1} \left(\mathbb{E}\left(X_2^2\right) - g_k(1) \right). \qquad (32)$$

When ψ is a linear combination of Hermite functions of order at most $k+1$, $\langle \psi, e_n \rangle = 0$ for $n \geq k+2$, causing all but the $k+1^{\text{th}}$ term to vanish, and equality is achieved:

$$\begin{aligned}
r_k(x) &= |x|^{k+1} r_k(1) \\
&= |x|^{k+1} \left(\mathbb{E}\left(X_2^2\right) - g_k(1) \right).
\end{aligned}$$

Note that as $g_k(1)$ increases monotonically with k, as its coefficients are all positive. Therefore, $\mathbb{E}(X_2^2) - g_k(1)$ will be decreasing, and thus we can loosen (32) slightly and write (where $n \leq k$)

$$r_k(x) \leq |x|^{k+1} \left(\mathbb{E}(X_2^2) - g_k(1)\right)$$

$$\leq |x|^{k+1} \left(\mathbb{E}(X_2^2) - g_n(1)\right).$$

We know, however, that $g_0(k) = \mathbb{E}(X_2)^2$, and therefore that

$$\begin{aligned} r_k(x) &\leq |x|^{k+1} \left(\mathbb{E}(X_2^2) - \mathbb{E}(X_2)^2\right) \\ &= |x|^{k+1} \mathrm{Var}(X_2), \end{aligned} \tag{33}$$

and thus, in order to achieve a remainder of at most $r_k(x)$, we must have

$$k \geq \frac{\log\left(r_k(x)/\mathrm{Var}(X_2)\right)}{\log x} - 1. \tag{34}$$

Hence we have rules of thumb enabling the determination of errors associated with the series truncation.

3.7 Numerical Error

The bound given in (31) is adequate only if we are to discount errors introduced by numerical integration. While in many cases this might be sufficient, we would do well to be able to predict the error introduced by our implementation.

Let the estimated value of each coefficient be denoted $b_n^\star$, with relative error at worst ε_r. That is, suppose $|b_n^* - |\langle\psi, e_n\rangle|| \leq \varepsilon_r |b_n^*|$. We have therefore an error that is at worst (by the triangle inequality)

$$\begin{aligned} |\Delta(x)| &\leq \sum_{n=0}^{k} |x|^n \left| {b_n^\star}^2 - |\langle\psi, e_n\rangle|^2 \right| \\ &\leq \sum_{n=0}^{k} |x|^n \left| b_n^\star + |\langle\psi, e_n\rangle| \right| \left| b_n^\star - |\langle\psi, e_n\rangle| \right| \end{aligned}$$

$$\begin{aligned} &\leq \varepsilon_r \sum_{n=0}^{k} |x|^n \, |b_n^*| \, |2b_n^* - (b_n^* - |\langle \psi, e_n \rangle|)| \\ &\leq \varepsilon_r \sum_{n=0}^{k} |x|^n |b_n^\star| \, (|2b_n^*| + \varepsilon_r |b_n^*|) \\ &= \left(\varepsilon_r^2 + 2\varepsilon_r\right) g_k \left(|x|\right). \end{aligned} \tag{35}$$

Noting that coefficient errors must be applied to (32), we add the truncation error to (35) and find that $g_k(r_i[k])$ differs from $r_o[k]$ by at most

$$|\epsilon(x)| \leq |x|^{k+1} \left[\mathbb{E}\left(X_2^2\right) - (1-\varepsilon_r)^2 g_k(1)\right] + \left(\varepsilon_r^2 + 2\varepsilon_r\right) g_k \left(|x|\right). \tag{36}$$

In the worst-case of $x = 1$, this reduces to

$$|\epsilon(1)| \leq \mathbb{E}\left(X_2^2\right) - (2\varepsilon_r^2 + 1) g_k(1)$$

and thus for a given worst-case error we have the requirement that

$$\varepsilon_r < \sqrt{\frac{1}{2}\left(\frac{\mathbb{E}(X_2^2) - |\epsilon(1)|}{g_k(1)} - 1\right)}. \tag{37}$$

4 Implementation

The implementation of the MNLT is now addressed. It has been implemented as a Matlab toolbox using a truncated power series for autocorrelation control. This task can be split into five main segments: the determination of $\eta(x)$, the calculation of the input autocorrelation function, production of the correlated Gaussian series, translation to the target distribution, and validation of the output. We address each of these individually in the following subsections.

Beyond this, the toolbox contains integrated documentation, and examples of its use are given in Appendix A.

4.1 Constructing the Transformation

The MNLT function $\eta(x) = F_{X_2}^{-1}(F_{X_1}(x))$, defined in Lemma 2.4, transforms a standard Gaussian random variable into one of the target distribution. However, it is clear that F_{X_2} is not analytically invertible in general. We thus require a numerical approximation of its inverse. This was done by sampling F_{X_2} uniformly over a bijective interval around $F_{X_2} = 0.5$. This then produced an approximation of the inverse using the method of cubic splines (see figure 4). The implementation is shown in Section B.1.

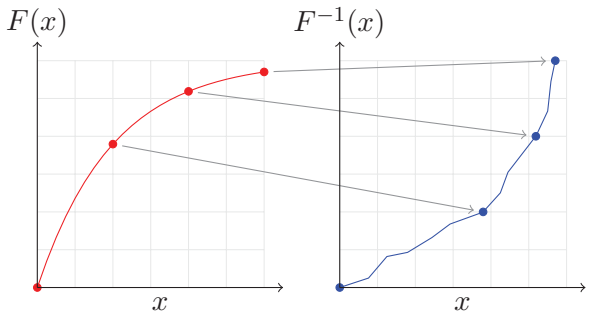

Figure 4: *Interpolation is used to determine values of the inverse CDF between samples.*

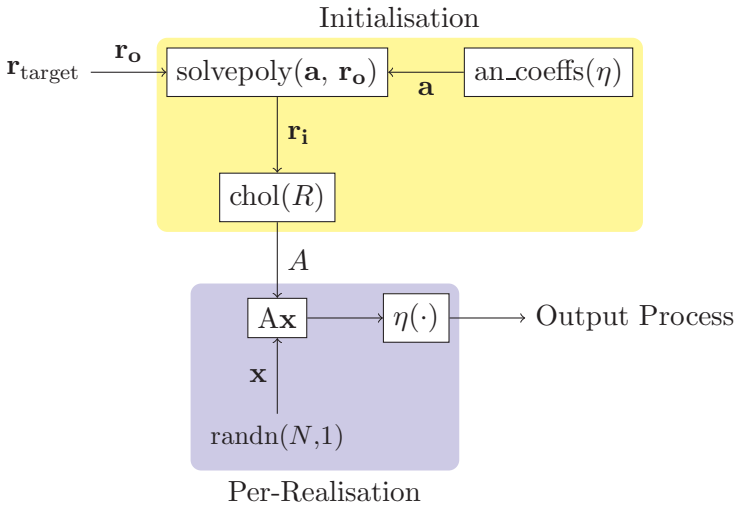

Figure 5: *The means by which the implementation correlates the output.*

4.2 Determining the Input Autocorrelation

We could use (5) to compute the values of $|\langle\psi, e_n\rangle|^2$, using the Monte Carlo approximation

$$|\langle\psi, e_n\rangle|^2 \approx \frac{1}{N}\sum_{n=1}^{N} \eta(x_n)\, H_n\left(\frac{x_n}{\sqrt{2}}\right), \tag{38}$$

where x_n are realisations of the standard normal distribution. This was used at first, however performance and accuracy were unsatisfactory.

Alternatively, we may use a deterministic method of numerical integration. The integral form of (5) can be written as

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)\, dx.$$

Such an integral can be estimated numerically via

$$\sum_{k=1}^{n} A_k f(x_k) + R(f), \tag{39}$$

where x_k are the roots of the n^{th} Hermite polynomial, and

$$A_k = \frac{2^{n+1} n! \sqrt{\pi}}{[H_n'(x_k)]^2}. \tag{40}$$

(See [24] for details of this). This formula is exact for the case when $f(x)$ is a polynomial of order at most $2n-1$. The remainder of this approximation is given by

$$R(f) = \frac{n!\sqrt{\pi}}{2^n} \frac{f^{(2n)}(c)}{(2n)!},$$

where c is the point at which $f^{(2n)}(x)$ takes its maximum.

A 22-point numerical integration using this method significantly outperformed the two-million point Monte Carlo simulation. However, difficulties in calculating (40) when using a large number of points prevented the use of this approximation when high accuracies were required.

Hence an adaptive technique was examined; specifically, the Gauss-Kronrod algorithm [22], whose implementation is included with Matlab. This provided a high degree of accuracy, and is thus used in our implementation.

Having done this, we now have a finite polynomial

$$g_k(r_{input}) = |\langle\psi, e_0\rangle|^2 + r_{input}|\langle\psi, e_1\rangle|^2 + r_{input}^2|\langle\psi, e_2\rangle|^2 + \cdots + r_{input}^k|\langle\psi, e_k\rangle|^2$$

which can be solved numerically.

Our next task in determining the autocorrelation of the input sequence is to find the desired autocorrelation of the output sequence. As we are using a truncation of (4), naïvely converting the autocovariance coefficient to autocorrelation using the known mean and variance of the target distribution will produce a non-standard Gaussian input. Whenever $g_k(x) \neq r_{output}(x)$, as in almost all cases, we have that

$$g_k(1) < r_{output}(1) = \mathbb{E}\left(X_2^2\right) = \mathrm{Var}(X_2) + \mathbb{E}\left(X_2\right)^2 \tag{41}$$

and hence solving $g_k(x)$ for the right-hand side of (41) would produce $r_{input}(0) \neq 1$.

This issue can be addressed by scaling the target autocovariance by a constant. Multiplying ρ by $(g_k(1) - \mathbb{E}(X_2)^2)/\mathrm{Var}(X_2)$ we instead find the equation

$$g_k(r_{input}) = \rho\left(g_k(1) - \mathbb{E}\left(X_2\right)^2\right) + \mathbb{E}\left(X_2\right)^2,$$

where ρ denotes the target autocovariance coefficient. In the case when $\rho = 1$, it is clear that $r_{input} = 1$ is indeed a valid solution. As $\rho = 1$ by necessity at zero-lag, we can then see that the resulting normal distribution will indeed be standard, and thus $\eta(X_1)$ will produce the correct output distribution.

4.3 Simulating the Correlated Process

In this brief section we clarify how the simulation of a correlated Gaussian process can be achieved. The key to this is the well-known Cholesky Decomposition of a covariance matrix [25]. Since a covariance matrix is positive definite, such a decomposition will always exists. This factorisation provides a matrix A such that if R is the covariance matrix, $R = AA^T$, where the latter matrix is the transpose of A.

The key idea in simulating a desired correlated Gaussian process is the following. Consider the effect of a linear transformation on a vector of independent Gaussian variables. Let $\mathbf{X} = [X_1 \cdots X_n]$ be a column vector of standard independent Gausssian variables, A be an $n \times n$ matrix, and $\mathbf{Y} = [Y_1 \cdots Y_n] = A\mathbf{X}$. Then it is relatively simple to show that Y is also a Gaussian process with covariance matrix $AA^T = R$. Hence, once the Cholesky factor matrix A is identified, we can simulate the correlated process using a matrix of independent and identically distributed standard Gaussian random variables and A. Specific to our simulation problem, we can generate a correlated vector $\mathbf{Z}$ in our target distribution with

$$\mathbf{Z} = \eta\left(A\mathbf{X}\right),$$

where $\eta(\mathbf{Y})$ is taken to be the vector $[\eta(Y_1) \cdots \eta(Y_n)]$.

4.4 Testing

Testing and validation of the simulator built to implement the MNLT was extensive. To test the capability of the simulator to generate uncorrelated processes, we used the Kolmogorov-Smirnov test [12, 23] to verify that the distribution of the output matched the theoretical distribution at a confidence level of $\alpha = 0.01$, using the Exponential and KK-Distributions with 10^5 elements in each realisation.

For correlated processes the Kolmogorov-Smirnov test is not an appropriate measure of the matching of probability distributions, because it only applies to realisations that are independent and identically distributed. Consequently a 50-bin χ^2 goodness of fit test [12] was used at a confidence level of $\alpha = 0.01$. To provide an adequate sample, the test was run against the concatenation of 1000 realisations, each of length 4000.

These confidence levels are relatively low as the tests are intended to be run repeatedly with various distributions as part of an automated test suite, and the use of $\alpha = 0.05$ is therefore likely to produce an unacceptable rate of false positives.

5 Simulation Results

We are now in a position to examine the results of the application of the MNLT to a number of clutter models of interest.

Recall that in Section 3.3 we found a closed-form relationship between the input and output autocorrelations of the MNLT when used to simulate the Log-Normal distribution. Figure 6 provides a comparison of the result of the MNLT with a series truncation performed, and the corresponding analytic results. The first plot in figure 6 shows the

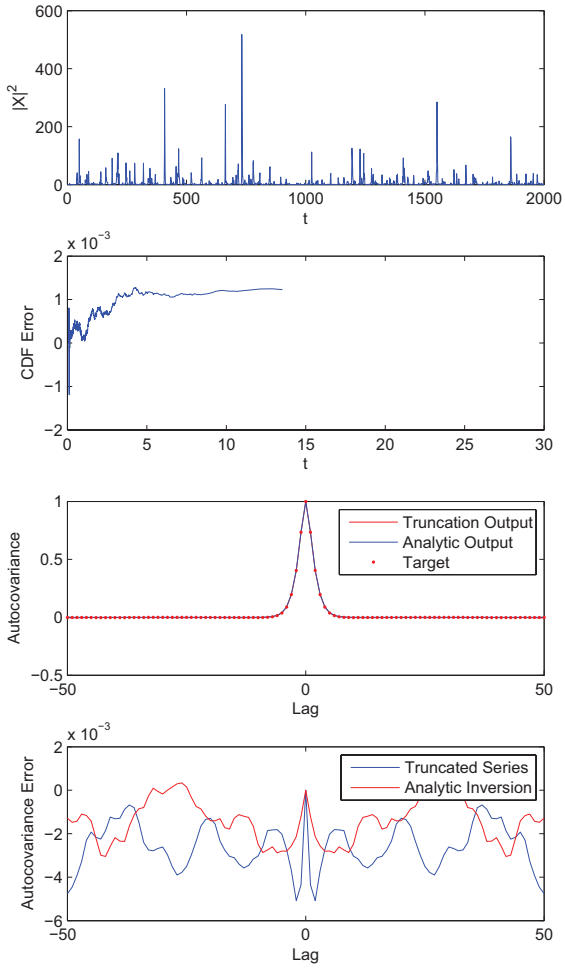

Figure 6: *Simulation of a Correlated Lognormal Process*

amplitude-squared simulation results, the second plot shows the error in cumulative distribution function, the third plot shows the autocovariance function and the fourth plot

shows the autocovariance error. The MNLT uses a fourth-order power series truncation. The distribution used here is the standard Lognormal $LN(0,1)$. We see that there is a peak error around zero in the approximate inversion, corresponding to larger target and input autocovariances. This error reached a maximum value of approximately 5×10^{-3}. It is clear here that the MNLT performs very well. Note that we are using an autocovariance coefficient given by the expression $r(k) = \frac{1+k}{e^k}$.

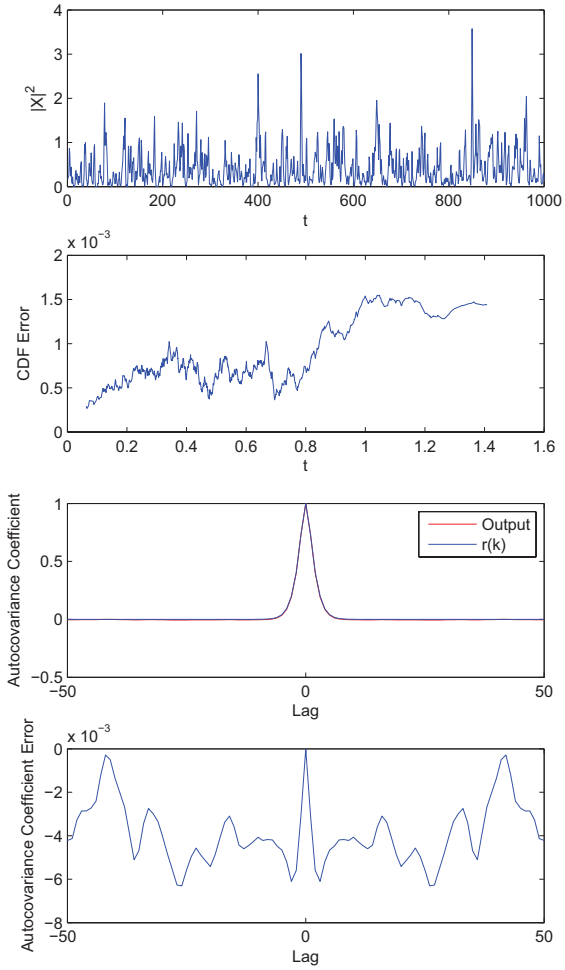

Figure 7: *Simulation of a Correlated Weibull Process*

Next we examine the case of generating correlated Weibull clutter; figure 7 shows the four relevant plots. The first shows the intensity measurements, the second shows the cumulative distribution function error, the third shows the autocovariance while the fourth is the corresponding error. In this simulation, the target distribution is the Weibull with cumulative distribution function $F(t) = 1 - e^{-2.4t^{1.9}}$. The correlation function is $r(k) =$

$\frac{1+k}{e^k}$. The output in plots three and four is the result of the MNLT processing of a fourth-order correlation function as before. We observe that the MNLT is working very well here as for the Lognormal example considered previously.

The next example we consider is for simulation of correlated KK-clutter returns. Recall that the KK-Distribution is the weighted sum of two K-Distributions; we can write its cumulative distribution function as

$$F_{KK}(t; k, \nu, c_1, c_2) = (1 - k)F_K(t; \nu, c_1) + kF_K(t; \nu, c_2),$$

where $0 \leq k \leq 1$ is the distribution mixing parameter, c_1 and c_2 are the two scale parameters and ν is the KK-Distribution's shape parameter. The individual K-Distributions used in the mixture have cumulative distribution function given by

$$F_K(t; \nu, c_i) = 1 - \frac{(c_i t)^\nu K_\nu(c_i t)}{2^{\nu-1}\Gamma(\nu)},$$

for $i \in \{1, 2\}$.

We consider the case where the KK-Distribution is characterised by $k = 0.01$, $\nu = 3.4119$, $c_1 = 110$ and $c_2 = 85$. The correlation function is $r(k) = \cos(0.1\pi k)e^{-0.1k}$. The first simulation is in figure 8, which examines the results of a linear approximation to the autocorrelation relationship. The autocovariance errors are quite large; this is perhaps to be expected, as it is the higher-order terms of (4) that encode the distortion produced by the MNLT.

The use of a quadratic approximation as shown in figure 9 improves dramatically the quality of the simulation, reducing the peak autocorrelation error to nearly one quarter that achieved through a linear approximation.

A fourth-order approximation is shown in figure 10, which further decreases error to approximately one half that achieved using a quadratic. It is clear that increasing the quality of the approximation will reduce the autocovariance error. Figure 11 demonstrates the simulation of another KK-distribution, showing that the simulator performs well for a wide range of parameters. In this case, we chose $k = 0.3, \nu = 2.5, c_1 = 1$ and $c_2 = 3$, as well as an autocovariance coefficient of $r(k) = (1 + t)e^{-k}$. We observe the MNLT again works very well for the KK-Distribution.

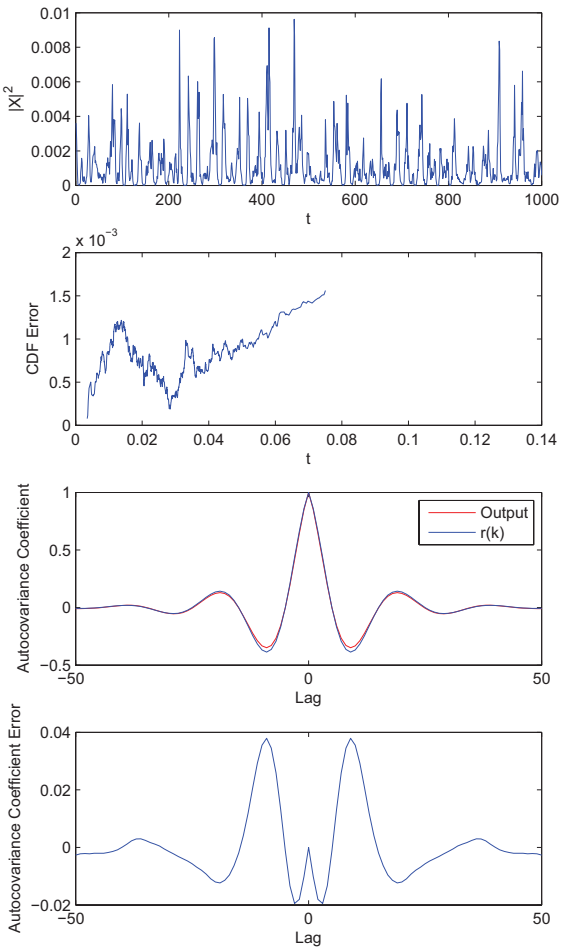

Figure 8: *Simulation of a KK Process With Linear Autocorrelation Correction*

Calculated values of the series coefficients and remainders are shown in Appendix C; we see that for the KK-Distribution, the series indeed converges quite rapidly.

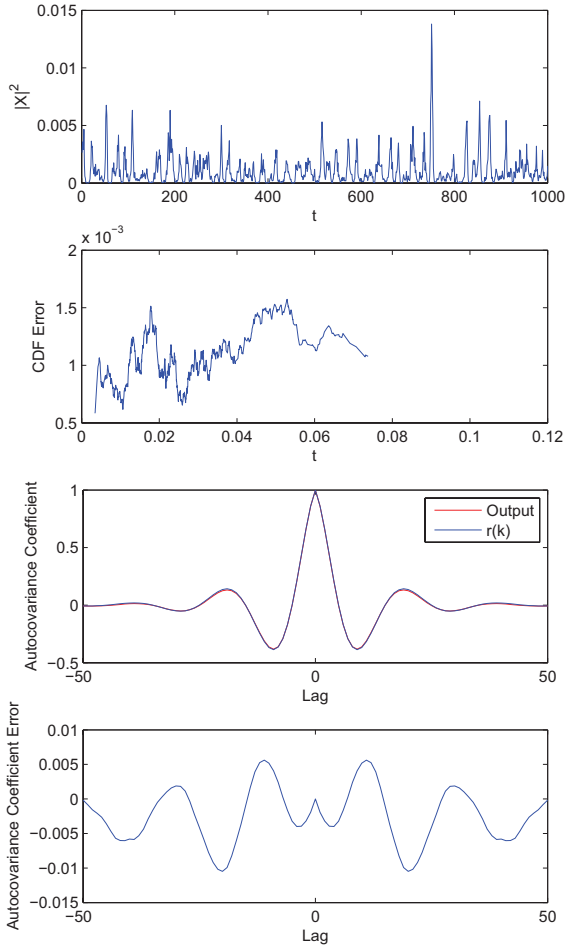

Figure 9: *Simulation of a KK Process With Quadratic Autocorrelation Correction*

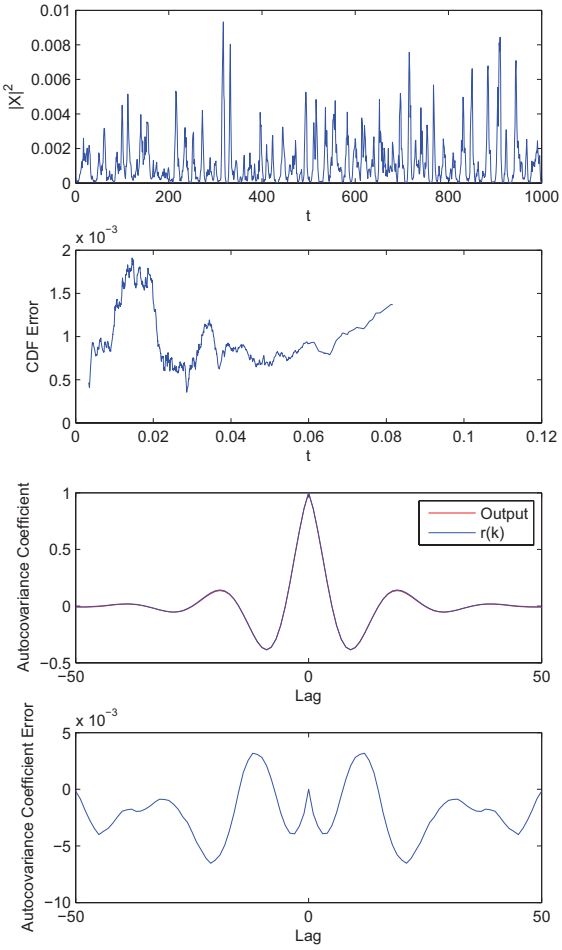

Figure 10: *Simulation of a KK Process with Quartic Autocorrelation Control*

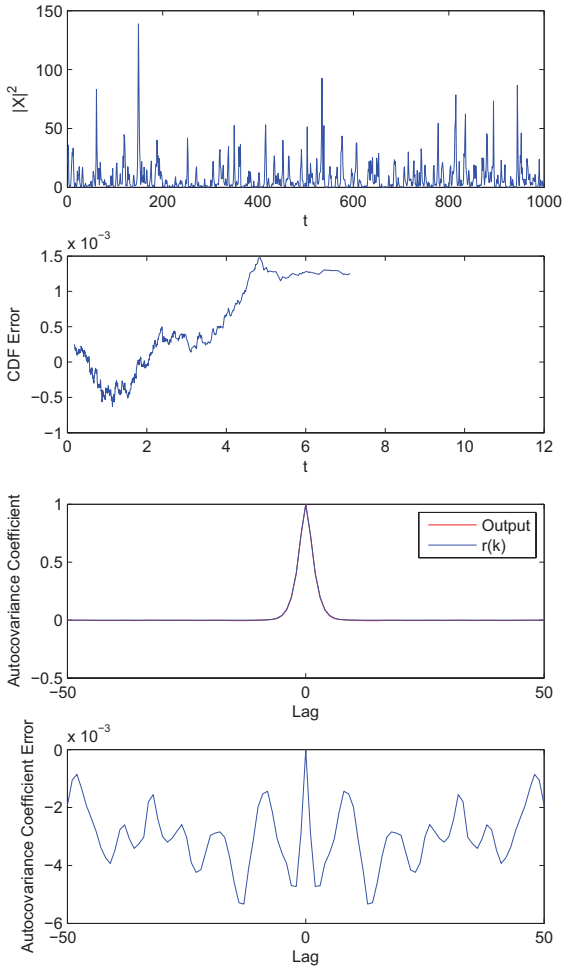

Figure 11: *Simulation of another KK Process with Quartic Autocorrelation Control*

6 Conclusions

The Memoryless Nonlinear Transform, which enables the simulation of correlated sea clutter from a prescribed distribution with given autocovariance function, has been introduced. This process involves converting a correlated Gaussian stochastic process (the input process) to a desired process with given autocovariance function (the output process). A new technique, which truncates the mapping between input and output process autocovariances and then performs an inversion, has been introduced. This enables the determination of which input Gaussian process should be used to generate the desired output stochastic process.

A simulator has been constructed in Matlab, and testing of some examples showed excellent results. In particular, we showed the MNLT works very well for earlier models of sea clutter, namely Lognormal and Weibull Distribution models. It was also demonstrated that the MNLT worked very well for the generation of correlated KK-Distributed clutter. Other tests, including an analysis for the case of the K-Distribution, further supported the validity of this simulator.

The simulator provides DSTO with a capability that can be integrated into radar models under development in the Microwave Radar Branch. This should prove to be useful for the longer term goals of testing high grazing angle detection schemes operating under correlated sea clutter returns.

Acknowledgements

The Co-author Lachlan Gunn expresses thanks to DSTO, and in particular EWRD, for providing the opportunity to be involved in its Summer Vacation Scholarship Program.

References

1. Tough, R. J. A. and Ward, K. D., 'The correlation properties of Gamma and other non-Gaussian processes generated by memoryless nonlinear transformation', *J. Phys. D: Appl. Phys.* **32**, 3075–3084, 1999.

2. Peebles, P. Z., 'The Generation of Correlated Log-Normal Clutter for Radar Simulations', *IEEE Trans. Areo. Elec. Sys.* **AES-7**, 1215–1217, 1971.

3. Farina, A., Russo, A. and Studer, F. A., 'Coherent radar detection in log-normal clutter', *IEEE Proc.***133**, 39–54, 1986.

4. Szajnowski, W. J., 'The generation of correlated Weibull clutter for signal detection problems', *IEEE Trans. AES***13**, 536–540, 1977.

5. Li, G. and Yu, K.-B., 'Modelling and Simulation of coherent Weibull clutter', *IEEE Proc.***136**, 2–12, 1989.

6. Ward, K. D., Tough, R. J. A. and Shepherd, P. W., 'Modelling Sea Clutter: Correlation, Resolution and Non-Gaussian Statistics', *IEE Radar 97*, Publication 449, 95–99, 1997.

7. Antipov, I., Simulation of Sea Clutter Returns. DSTO-TR-0679, 1998.

8. Ross, S., Stochastic Processes. (Wiley, New York, 1983).

9. Durrett, R., Probability: Theory and Examples, (Wadsworth Publishing Company, 1986).

10. Billingsley, P., Probability and Measure, (John Wiley and Sons, 1986).

11. Ross, S. M., Simulation, (Academic Press, San Diego, California, 2002).

12. Shao, J., Mathematical Statistics (Springer, 2003)

13. Levanon, N., Radar Principles, (Wiley, New York, 1988).

14. Skolnik, M., Introduction to Radar Systems: Third Edition, (McGraw-Hill, New York, 2008).

15. Stein, S. and Storer, J. E., 'Generating a Gaussian Sample', *IRE Trans. Info. Theory* **2**, 87–90, 1956.

16. Kreyszig, E., Introductory Functional Analysis with Applications (Wiley, New York, 1978).

17. Harper, C., Analytic Methods in Physics, (Wiley, Berlin, 1999).

18. Renardy, M. and Rogers, R. C., An Introduction to Partial Differential Equations, (Springer, New York, 2004)

19. Nieto, M. N. and Truax, D. R., 'Arbitrary-order Hermite generating functions for obtaining arbitrary-order coherent and squeezed states', *Phys. Lett. A* **208**, 8–16, 1995.

20. Dong, Y., Distribution of X-Band High Resolution and High Grazing Angle Sea Clutter. DSTO-RR-0316, 2006.

21. Li, S. T. and Hammond, J. L., 'Generation of Pseudorandom Numbers with Specified Univariate Distributions and Correlation Coefficients' *IEEE Trans. Syst., Man, Cybern* **5**, 557–561, 1975.

22. Kahaner, D., Moler, C. and Nash, S. 'Numerical Methods and Software', (Prentice-Hall, 1989).

23. Shorak, G.R. and Wellner, J.A., 'Empirical Processes with Applications to Statistics', (Wiley 1986).

24. Krylov, V. I., and Stroud, A.H., 'Approximate Calculation of Integrals' (Macmillan, 1962). Translation of 'Priblizhennoe Vychislenie Integralov', Krylov, V. I., 1959.

25. Hogben, L. 'Handbook of Linear Algebra' (Chapman & Hall/CRC, 2007)

Appendix A Examples of Toolbox Usage

This section describes some basic use of the toolbox. More detailed documentation, including a tutorial and a short explanation of the theory of operation, is available in the Matlab help browser and in the help for individual functions (see Section B).

A.1 Generating an Uncorrelated Series

This example shows how one might simulate a uncorrelated exponential process. Using the *expcdf* function from the Matlab Statistics Toolbox, it produces one million uncorrelated samples distributed according to $Exp(1/2)$. The code in Listing 1 is also available in the toolbox as *example/uncorrelated.m*

Listing 1: *Uncorrelated Simulation*

```
% Generate an uncorrelated exponential distribution, and show
% some features of the process as well as a histogram.

ctx = Simulator(@(t) expcdf(t,2), 1e6, 0.01);
X = simulate(ctx);

fprintf('Output size:\n');
disp(size(X));

fprintf('Output mean:\n');
disp(mean(X));

fprintf('Output variance:\n');
disp(var(X));

hist(X, 1000);
```

Listing 2: *Uncorrelated Simulation Output*

```
Output size:
            1       1000000

Output mean:
     2.0011

Output variance:
     4.0025
```

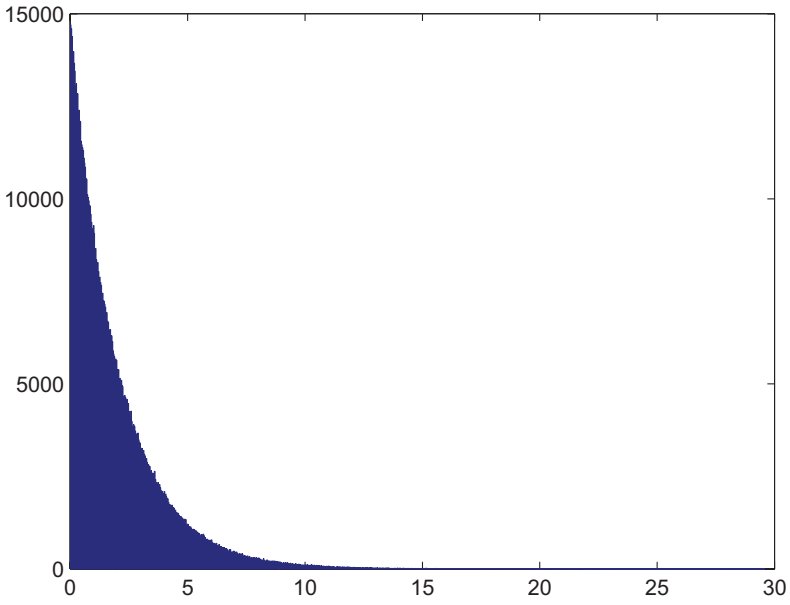

Figure A1: *Uncorrelated Simulation Output Histogram*

A.2 Generating a Correlated Series

This example shows how one might simulate an correlated exponential process. The interface is the same as the uncorrelated case, however memory and computational considerations restrict us to a shorter sequence. The code in Listing 3 is also available in the toolbox as *example/correlated.m*.

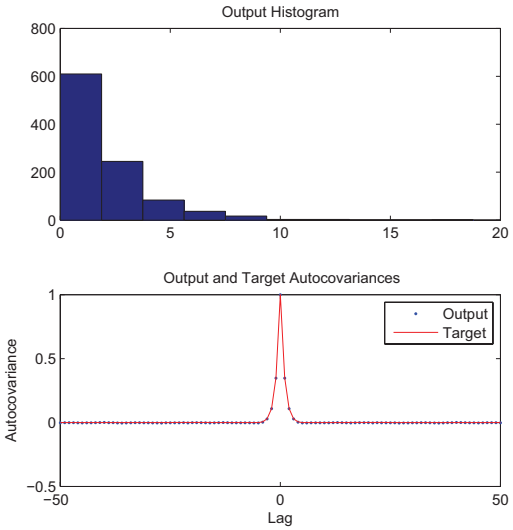

Figure A2: *Correlated Simulation Output*

Listing 3: *Correlated Simulation*

```
% Generate a correlated exponential distribution, and show
% some features of the process as well as a histogram and autocovariance
% plot.

r = @(t) cos(t*pi/10).*exp(-t);

ctx = Simulator(@(t) expcdf(t,2), r(0:999), 0.01);
X = simulate(ctx);

subplot(2,1,1);
hist(X);
title('Output Histogram');

% Average the autocovariance over 1k realisations.
lags = -50:50;
cov = zeros(size(lags));

for i = 1:1000;
    X = simulate(ctx);
    cov = cov + xcov(X, 50, 'coeff');
end;
cov = cov / 1000;

subplot(2,1,2);
scatter(lags, cov, '.b');
hold on; plot(lags,r(abs(lags)),'r'); hold off;
box on;

legend('Output', 'Target');
xlabel('Lag');
ylabel('Autocovariance');
title('Output and Target Autocovariances');
```

Appendix B Simulator Toolbox Code Listings

```
function obj = Simulator(F, r, varargin)
    % Simulator  Simulate an autocorrelated time series.
    %
    %   ctx = Simulator(F, N);
    %   ctx = Simulator(F, autocovariance(1:N));
    %
    %   ctx = Simulator(F, r, w);
    %
    %   ctx = Simulator( ..., 'approximationOrder', n);
    %   ctx = Simulator( ..., 'approximationTol', n);
    %   ctx = Simulator( ..., 'controlAutocorrelation', c);
    %
    % Example:
    %
    %   % Generate uncorrelated exponential variables.
    %   F      = @(t) ( 1-exp(-t) ) .* (t >= 0);
    %   ctx    = Simulator(F, 1000);
    %
    %   X_1    = simulate(ctx);
    %   X_2    = simulate(ctx);
    %
    %   % Generate correlated exponential variables.
    %   F      = @(t) ( 1-exp(-t) ) .* (t >= 0);
    %   r      = @(t) cos(t*pi/10).*exp(-t/10);
    %   ctx    = Simulator(F, r(0:1000));
    %   X      = simulate(ctx);
    %
    % A simulator object can be used to generate autocorrelated
    % sequences of arbitrary distributions.  Each instance
    % is immutably tied to a distribution, sequence length and
    % autocovariance.
    %
    % The simulate method produces a realisation of
    % an autocorrelated time series of the distribution with the
```

```
% given CDF.
%
% When called with a vector r of length N, correlated vectors
% of length N will be generated with the provided
% autocorrelation coefficients (ie. r(1) should be equal to one).
%
% When called with a scalar N in place of an autocovariance vector,
% uncorrelated vectors of length N will be generated.
%
% The w argument determines the spacing between samples of the
% CDF.  If unspecified, this defaults to 0.01.  Set this to a smaller
% value for CDFs that change quickly.
%
% The approximationOrder argument determines the order of
% the polynomial approximation used when controlling the
% autocovariance.  If not specified, this defaults to four.
%
% The approximationTol argument specifies the relative tolerance of
% the autocorrelation approximation coefficients.
% This defaults to 1e-9.
%
% The controlAutocovariance argument disables output
% autocovariance control.  If this is false, then the input
% autocovariance will be used directly with the MNLT.  This
% might be useful in cases where the effect of the MNLT on
% the output autocovariance can be determined analytically.
%
% See also Simulator.simulate

% Initial values to make sure the fields are created.
obj.U = [];
obj.correlated = 0;
obj.variance = 1;

obj.approximationOrder = 0;
```

```
obj.approximationTol    = 0;
obj.controlled = false;
obj.w = 0;
obj.F = 0;
obj.Finv = 0;

% Produce a dummy class if called with no arguments so that we
% can load Simulator objects from a file.
if nargin == 0
    obj = class(obj, 'Simulator');
    return;
end

p = inputParserCompat();
p = addRequired(p, 'F', @(x) isa(x, 'function_handle') );
p = addRequired(p, 'r', @isnumeric );
p = addOptional(p, 'w', 0.01, @(x) isnumeric(x) && isscalar(x) && x > 0);
p = addParamValue(p, 'controlAutocovariance', 1, @(t) t==0||t==1);
p = addParamValue(p, 'approximationOrder', 4, @(t) 0 == rem(t,1) );
p = addParamValue(p, 'approximationTol', 1e-9, @(t) 0 < t && t < 1);
p = parse(p,F,r,varargin{:});

results = Results(p);

obj.approximationOrder = results.approximationOrder;
obj.approximationTol   = results.approximationTol;
obj.controlled         = results.controlAutocovariance;
obj.w                  = results.w;
obj.F = F;

% We begin by inverting the CDF.
[a b] = cdf_bounds(F, obj.w);
obj.Finv = inversef(F, a, b, obj.w);

r_size = size(r);
```

```
if isscalar(r)
    % We have been given a scalar, and so generate
    % uncorrelated values.  Since we need no correlation
    % matrix, we save memory and only store the number of
    % values to generate.

    obj.U = r;
    obj.correlated = 0;
else
    % We have an autocorrelation function/matrix, and so
    % generate correlated data.

    % If the output is to be correlated, then we must
    % provide an determine the input covariance that
    % is needed to produce the desired output correlation.
    if obj.controlled
        r_input = invert_autocorrelation(r, obj.Finv, ...
                                         obj.approximationOrder, ...
                                         obj.approximationTol);
    else
        % We do not need to control the autocovariance.
        r_input = r;
    end

    obj.variance = r_input(1);

    obj.correlated = 1;
    if isvector(r)
        % We have been given a vector, implying that the
        % process is stationary.

        % If the process is stationary then we must construct
        % a correlation matrix.
        %
```

```
        % DOC_GAUSSIAN_GENERATE
        C = ones(length(r));
        for i = 1:(length(r))
            for j = 1:(length(r))
                C(i,j) = r_input(abs(i-j) + 1);
            end
        end

        % We use a Cholesky factorisation so that by
        % multiplying a vector of normal variables by U we
        % produce a vector of Gaussian variables with the given
        % correlations.
        [obj.U p] = chol(C);
        % Check whether C was positive-definite.
        if p == 1
            error('Simulator:Construct:BadCovariance',...
                    ['Invalid autocovariance; '...
                    'Gaussian covariance matrix must ' ...
                    'be positive definite.']);
        end


    elseif r_size(1) == r_size(2)
        % TODO: Simulate processes with nonstationary autocovariances.
        error('Simulator:Construct:NotImplemented', ...
          'Non-stationary correlations not implemented.');
    else
        error('Simulator:Construct:NonSquareMatrix', ...
         'Autocorrelation must be scalar or vector.');
    end

end

obj = class(obj, 'Simulator');
```

```
end
```

B.1 CDF Inversion

```
function [a b] = cdf_bounds(F, w)
    % CDF_BOUNDS  Determine CDF bounds of inversion.
    %
    % The bounds are discovered by placing both
    % at a point near F^-1(0.5), then decreasing and
    % increasing the lower and upper bounds respectively
    % until we reach a point at which moving further results
    % in no change.

    % Arbitrary initial starting value.
    a = 0;

    % We first locate ourselves at approximately the median.
    while (F(a+w) < 0.5)

        n = 0;
        while F(a+w*2^n) < 0.5
            n = n + 1;
        end
        a = a + w*2^n;
    end

    while (F(a) > 0.5)
        n = 0;
        while F(a-w*2^n) > 0.5
            n = n + 1;
        end
        a = a - w*2^n;
    end

    b = a;

    % Then, we expand the upper and lower bounds outwards until
    % either we reach an invalid value or the limit of numerical
```

```
    % resolution.
    %
    % The goal here is that in the range over which we invert F,
    % the inverse is a single-valued.
    while (F(a-w) >= 0 && F(a) ~= F(a+w))
        a = a - w;
    end

    while (F(b+w) <= 1 && F(b) ~= F(b+w))
        b = b + w;
    end

    % Some safety margin.
    a = a + w;
    b = b - w;
end

function f = inversef(F, a, b, width)
    % INVERSEF Approximate an inverse CDF.
    %
    % The inversef function approximates an inverse by sampling
    % the function between the bounds given at points separated
    % by the width argument.  It then interpolates between
    % these points to return a function handle based on the
    % piecewise polynomial generated by the interpolator.
    %
    % The function returned will be equal to a at points less than
    % F(a), and b at points greater than F(b0.
    %
    % See also interp1.

    x = a:width:b;
    y = F(x);

    % Generate a piecewise polynomial from our samples.
    pp = interp1(y,x,'spline','pp');
```

```
    % Construct our function handle.
    f = @(t) ppval(pp, t).*( (t >= F(a)).*(t <= F(b)) ) + ...
                (t < F(a))*a + (t > F(b))*b;
```

end

B.2 Correlation Control

```
function b_n = an_coeffs(eta, max_n, reltol)
    % AN_COEFFS Determine coefficients for the autocorrelation series.
    %
    % The an_coeffs takes as input a function handle eta, and returns
    % the first max_n coefficients using an adaptive quadrature.

    n = fliplr(0:max_n);

    b_n = zeros(max_n+1, 1);
    first = true;
    smallest_place = 1;
    for k = n;
        H = hermite_poly(k);

        integrand = @(t) zero_infinities( ...
                            polyval(H/sqrt(factorial(k)*2^k), t) ...
                        .* exp(-t.^2) .* eta(t*sqrt(2)));
        b_n(max_n-k+1) = quadgk(integrand,-Inf,Inf, ...
                                'RelTol',reltol)^2/pi;
    end

function r_input = invert_autocorrelation(r, Finv, approximationOrder, ...
                                                    tol)
    % INVERT_AUTOCORRELATION  Invert the I->O autocorrelation relationship.
    %
    % The invert_autocorrelation function determines the input
    % autocorrelation of the standard Gaussian input to produce the desired
    % output autocorrelation using the method derived by
    % Tough and Ward (1999).
    %
    % Numerical integration is used to determine the first few coefficients
    % of the power series relating the autocorrelations.  The provided
    % autocovariance coefficients are converted to our desired output
    % autocorrelation, and the polynomial made by truncating the power
```

```
    % series is solved, yielding our output coefficients.

    autocorrelation_poly = an_coeffs(@(t) Finv(0.5*(1+erf(t/sqrt(2)))), ...
                                             approximationOrder, ...
                                             tol);

    % We have been given autocovariance coefficients, but in fact need
    % an autocorrelation.  Fortunately, we can convert between
    % them.
    %
    % While E[X_a X_(a-k)] = r[k] * var(X) + mean(X)^2, we must have
    % r_input[0] = 1 (which our approximation does not preserve), and thus
    % scale r[k] by a constant to yield this autocorrelation, whose
    % inversion at zero *is* equal to one.
    mean_squared = autocorrelation_poly(length(autocorrelation_poly));
    r = r*(polyval(autocorrelation_poly,1) - mean_squared) ...
        + mean_squared;

    % All this done, we can perform the inversion itself.
    r_input = zeros(size(r));
    for i = 1:length(r)
        r_input(i) = solvepoly(autocorrelation_poly, r(i));
    end
end

function x = solvepoly(p, r)
        % SOLVEPOLY Solve a polynomial equated to a constant.
        %
        % The solvepoly function solves equations of the form
        %
        %   r = a_n x^n + ... + a_1 x + a_0
        %
        % returning the greatest real solution.

        % This is equivalent to an equation of the form
        %
```

```
    %    0 = a_n x^n + ... + a_0 - r
    %
    % which is already solvable.
    new_p = p;
    new_p(length(p)) = new_p(length(p)) - r;

    candidate_solutions = roots(new_p);

    % We restrict ourselves to real solutions.
    real_solutions = imag(candidate_solutions) == 0;
    candidate_solutions = candidate_solutions(real_solutions);

    % We select the greatest of these solutions.
    x = max(candidate_solutions);
end
```

B.3 Series Generation

```
function CorrelatedN = generate_gaussian(obj)
    % GENERATE_GAUSSIAN  Generate correlated Gaussian variables.
    %
    % The generate_gaussian method produces a vector of correlated
    % standard Gaussian variables based upon the correlation
    % characteristics specified during initialisation.
    %
    % The vector generated will be of the size of the correlation
    % sequence, the rank of the correlation matrix, or the size
    % given during initialisation, as appropriate.
    %
    % Example:
    %
    %   ctx = Simulator_OldStyle(F, r, w);
    %   X   = generate_gaussian(ctx);

    if obj.correlated == 0
        CorrelatedN = randn(1, obj.U);
    else
        normal_vars = randn(1, length(obj.U));
        CorrelatedN = normal_vars * obj.U;
    end
end

function [CorrelatedU CorrelatedN] = generate_uniform(obj)
    % GENERATE_UNIFORM  Generate correlated uniform variables.
    %
    % The generate_uniform method produces a vector of correlated
    % uniform variables on [0,1] by manipulating a vector of
    % correlated normal variables.
    %
    % Example:
    %
    %   ctx = Simulator_OldStyle(F, r, w);
```

```
    %   X    = generate_uniform(ctx);

    CorrelatedN = generate_gaussian(obj);
%     CorrelatedU = normcdf(CorrelatedN);
    CorrelatedU = 0.5*erfc(CorrelatedN/sqrt(2));
end

function [X G U] = simulate(obj)
    % SIMULATE  Generate correlated realisation.
    %
    % The simulate method produces a realisation of
    % an autocorrelated vector of the distribution with the
    % given CDF.
    %
    % Before calling simulate, the CDF must be set with the
    % 'F' attribute.
    %
    % Example:
    %
    %   F    = @(t) ( 1-exp(-t) ) .* (t >= 0);
    %   ctx = Simulator_OldStyle(F, 1000, 0.01);
    %   X    = simulate(ctx);
    %
    % In addition to the realisation of the desired distribution,
    % the method returns the corresponding normal and uniform
    % vectors that were used to generate the distribution.
    %
    %   [X gaussian uniform] = ctx.simulate();
    %
    % The relationship between these is such that
    %
    %   eta(gaussian) = X
    %
    % and thus this is sufficient to tune the input autocovariance.
```

```
    % We proceed by applying the inverse CDF to the uniform
    % distribution.
    [U G] = generate_uniform(obj);
    % We must take the real part, as otherwise we sometimes get
    % x + 0j, which Octave does not like.
    X = real(obj.Finv(U));
end
```

B.4 Miscellaneous

```
function [H Hnm1] = hermite_poly(N)
    % HERMITE_POLY Return the n'th Hermite Polynomial.
    %
    % The hermite_poly function returns the coefficients of the n'th
    % Hermite Polynomial as the first value, and the coefficients
    % of the n-1'th Hermite Polynomial, with a leading zero.
    %
    % The nonzero coefficient of greatest order for the polynomial of
    % order n is 2^n.

    if N == 0
        H    = [1];
        Hnm1 = [];
    elseif N == 1
        H    = [2  0];
        Hnm1 = [1];
    else
        Hnm1 = [1];
        H = [2  0];

        for i = 1:(N-1)
            old_H = H;
            H = [H*2  0] - [0  0  Hnm1*2*i];
            Hnm1 = old_H;
        end
```

```
        Hnm1 = [0 Hnm1];
    end
end
```

B.5 InputParser Replacement

The Matlab inputParser class was required, but this was not available in Octave. We have written our own, included here.

```
function obj = inputParserCompat(ip)
    if nargin == 0
        obj.required = {};
        obj.optional = {};
        obj.params   = {};
        obj.Results  = {};

        obj = class(obj, 'inputParserCompat');
    elseif strcmp(class(ip), 'inputParserCompat')
        obj = ip;
    else
        error('inputParserCompat: Expecting no arguments.');
    end
end

function obj = addRequired(this, name, validator)

    obj = this;

    param.name = name;
    param.validator = validator;

    id = length(obj.required)+1;
    obj.required{id} = param;

end

function obj = addOptional(this, name, default, validator)
```

```
    obj = this;

    param.name = name;
    param.default = default;
    param.validator = validator;

    id = length(obj.optional)+1;
    obj.optional{id} = param;

end

function obj = addParamValue(this, name, default, validator)

    obj = this;

    param.name = name;
    param.default = default;
    param.validator = validator;

    id = length(obj.params)+1;
    obj.params{id} = param;

end

function obj = parse(this, varargin)

    obj = this;

    if nargin < length(obj.required)+1
        error('Not enough parameters.');
    end

    % We start by dealing with all of the required arguments.
    i = 1;
    for param = obj.required
```

```
        value = varargin{i};
        param = param{1};
        if isa(param.validator, 'function_handle')
            if ~param.validator(value)
                error(['Parameter ''' param.name ''' failed validation.']);
            end
        end

        obj.Results = setfield(obj.Results, param.name, value);

        i = i + 1;
    end

    % Now set the optional arguments that were supplied.
    parameter = 1;
    while i <= length(varargin) && parameter <= length(obj.optional)
        value = varargin{i};
        param = obj.optional{parameter};

        if isa(param.validator, 'function_handle')
            if ~param.validator(value)
                error(['Parameter ''' param.name ''' failed validation.']);
            end
        end

        obj.Results = setfield(obj.Results, param.name, value);

        i = i + 1;
        parameter = parameter + 1;
    end

    % Set the other optional arguments to their default values.
    while parameter <= length(obj.optional)
        param = obj.optional{parameter};
        obj.Results = setfield(obj.Results, param.name, param.default);
```

```
    parameter = parameter + 1;
end

% Set all of the named arguments to their default values.
for param = obj.params
    param = param{1};

    obj.Results = setfield(obj.Results, param.name, param.default);
end

% Now look for the remaining named arguments.
while i < length(varargin)
    paramName = varargin{i};
    if ~ischar(paramName)
        error('Parameter name must be a string.');
    end

    matched = 0;
    for param = obj.params
        param = param{1};
        if strcmp(paramName, param.name)

            value = varargin{i+1};

            if isa(param.validator, 'function_handle')
                if ~param.validator(value)
                    error(['Parameter ''' param.name ...
                            ''' failed validation.']);
                end
            end

            obj.Results = setfield(obj.Results, param.name, value);

            matched = 1;
```

```
                break;
            end
        end

        if matched == 0
            error(['''' paramName ''' is not a parameter.']);
        end

        i = i + 2;
        if i == length(varargin)
            error('No name specified for parameter.');
        end
    end

end

function r = Results(obj)
    r = obj.Results;
end

function disp(obj)
    fprintf('Required:\n');
    disp(obj.required);

    fprintf('Optional:\n');
    disp(obj.optional);

    fprintf('Named:\n');
    disp(obj.params);
end
```

B.6 Testing

```
function fail = test_distribution(id, F, samples, w, dist_alpha, type)
    % TEST_DISTRIBUTION Test the Simulator's uncorrelated distribution.
    %
    % The test_distribution function uses Kolmogorov–Smirnov to test the
    % output distribution.
    %
    % Use type = 1 when using with xUnit.
    ctx = Simulator(F, samples, w);

    fail = 0;
    if type == 0
        fprintf('Test %d (uncorrelated simulation)\n', id);
        fprintf('\tKolmogorov–Smirnov.....');
    end

    X = simulate(ctx);
    min_x = min(X);
    max_x = max(X);

    % If we are using Matlab, then we use the builtin kstest function.
    if ~exist('OCTAVE_VERSION', 'builtin')
        cdf_points = min_x:( (max_x–min_x)/100 ):max_x;

        [dist_fail p] = kstest(X, [cdf_points; F(cdf_points)]', ...
                               dist_alpha);
    else
        % With Octave, things are somewhat more tricky. We need a function
        % entitled <name>_cdf. As the distribution can change, we have
        % made such a function that calls the function–handle in a global
        % variable id_cdf_f, allowing the builtin Kolmogorov–Smirnov to
        % work.

        % If id_cdf_f is already set, then save its current value and clear
        % it.
```

```
if exist('id_cdf_f', 'variable')
    old_id_cdf_exists = true;
    old_id_cdf_value  = id_cdf_f;

    if ~isempty(whos('global','variable'))
        old_cdf_global = true;
    else
        old_cdf_global = false;
    end

    clear id_cdf_f;
else
    old_id_cdf_exists = false;
end

% Set our CDF and run K-S on the dataset.
global id_cdf_f;
id_cdf_f = F;

p = kolmogorov_smirnov_test(X, 'identity');
if p < dist_alpha
    dist_fail = true;
else
    dist_fail = false;
end

% Restore id_cdf_f to its old value.
if old_id_cdf_exists
    clear id_cdf_f;

    if old_cdf_global
        global id_cdf_f;
    end

    id_cdf_f = old_id_cdf_value;
```

```
        end
    end

    % Print output or provide success/failure status to xUnit.
    if type == 0
        if dist_fail == 1
            fail = fail + 1;
            fprintf('failed. (p = %f)\n\n', p);
        end

        if fail == 0;
            fprintf('passed.\n\n');
        end
    else
        message = sprintf( ...
                'Kolmogorov–Smirnov failed at level alpha = %f',...
                dist_alpha);

        assertEqual(dist_fail, false, message);
    end
end

function fail = test_correlation(id, F, r, samples, realisations, ...
                                 maxlag, w, ...
                                 cdf_alpha, ...
                                 cdf_error_threshold, ...
                                 covariance_error_threshold, ...
                                 test_type)
    % TEST_CORRELATION  Test the Simulator with autocorrelated data.
    %
    % Use test_type = 1 when testing with xUnit.

    ctx = Simulator(@(t) F(t), r(0:(samples-1)), w);

    if test_type == 0
        fail = 0;
```

```
    fprintf('Test %d (correlated simulation)\n', id);
end

X = simulate(ctx);
if exist('OCTAVE_VERSION', 'builtin')
    cdf_points = unique(X);
    cdf_values = empirical_cdf(cdf_points, X);
else
    output  = zeros(1, samples*realisations);
    output(1:samples) = X;
end


% We need to store the output covariances (so that we can test for
% autocovariance error) and the output itself (so that we can run
% the chi-square test for the distribution.
cov_out = zeros(size(-maxlag:maxlag));

for i = 1:realisations
    X   = simulate(ctx);

    if exist('OCTAVE_VERSION')
        cdf_values = cdf_values + empirical_cdf(cdf_points, X);
    else %Matlab

        % Save our current realisation for later.
        output( (samples*i+1):(samples*(i+1)) ) = X;

    end


    cov_out = cov_out + xcov(X, maxlag, 'coeff');
end

lags = -maxlag:maxlag;
```

```
cov_out  = cov_out / realisations;

% If we are using Octave, we look at the difference between actual
% and empirical CDF, as there is no Chi-square goodness of fit built
% in.
if exist('OCTAVE_VERSION', 'builtin')

    max_cdf_error = max(abs(cdf_values/realisations - F(cdf_points)));
    if max_cdf_error > cdf_error_threshold
        cdf_fail = true;
        cdf_error_text = sprintf(' (max error %f)', max_cdf_error);
    else
        cdf_fail = false;
    end

else
    % Matlab has a Chi-Square goodness-of-fit test built in, so we use
    % this instead.

    [cdf_fail p] = chi2gof(output, 'cdf', F, ...
                                   'nbins', 10, ...
                                   'alpha', cdf_alpha);
    cdf_error_text = sprintf(' (p = %f)', p);

    cdf_fail = logical(cdf_fail);

end

% We test the error by simply comparing the autocovariance coefficients
% output with the target.
ac_error = abs(cov_out - r(abs(lags)));
max_ac_error = max(ac_error);

% If we are not using xUnit, print the test output ourselves.
if test_type == 0
```

```
            fprintf('\tCDF error ..............');
            if cdf_fail
                fail = fail + 1;
                fprintf('failed.%s\n', cdf_error_text);
            else
                fprintf('passed.\n');
            end

            fprintf('\tAutocovariance error...');
            if max_ac_error > covariance_error_threshold
                fprintf('failed. Greatest error %f.\n\n', ...
                    max_ac_error, covariance_error_threshold);

                fail = fail + 1;

                return;
            else
                fprintf('passed.\n\n');
            end
        else
            % Otherwise call xUnit functions.

            assertFalse(cdf_fail, 'Distribution test failed.');
            assertTrue(max_ac_error < covariance_error_threshold, ...
                        'Autocovariance error too large.')
        end

    end
```

Appendix C Tables of Numerical Results

We have made numerical estimates for the coefficients of (26) for several distributions.

	Exp(1)		K($c=1,\nu=1$)	
n	$\lvert\langle\psi,e_n\rangle\rvert^2$	$r_n(1)$	$\lvert\langle\psi,e_n\rangle\rvert^2$	$r_n(1)$
0	1.000000	1.000000	2.467396	1.532604
1	0.815765	0.184235	1.339585	0.193019
2	0.177391	0.006844	0.188737	0.004282
3	0.006685	0.000159	0.004139	0.000143
4	0.000134	0.000025	0.000085	0.000058
5	0.000017	0.000008	0.000004	0.000054
6	0.000007	0.000001	0.000001	0.000054

Table C1: *Estimates of* $|\langle\psi,e_n\rangle|^2$ *for the Exponential and* $\mathrm{K}(1,1)$ *distributions.*

	K($c=1,\nu=2$)		K($c=1,\nu=4$)	
n	$\lvert\langle\psi,e_n\rangle\rvert^2$	$r_n(1)$	$\lvert\langle\psi,e_n\rangle\rvert^2$	$r_n(1)$
0	5.551652	2.448348	11.806900	4.193100
1	2.235475	0.212873	3.939934	0.253167
2	0.210615	0.002258	0.252644	0.000522
3	0.002197	0.000060	0.000485	0.000037
4	0.000054	0.000006	0.000024	0.000013
5	0.000000	0.000006	0.000011	0.000002
6	0.000000	0.000006	0.000002	0.000001

Table C2: *Estimates of* $|\langle\psi,e_n\rangle|^2$ *for the* $\mathrm{K}(1,2)$ *and* $\mathrm{K}(1,4)$ *distributions.*

	N($\mu = 1, \sigma = 2$)		LN($\mu = 0, \sigma = 1$)	
n	$\lvert\langle\psi, e_n\rangle\rvert^2$	$r_n(1)$	$\lvert\langle\psi, e_n\rangle\rvert^2$	$r_n(1)$
0	1.000000	4.000000	2.71828	4.67077
1	4.000000	0.000000	2.71828	1.95249
2	0.000000	0.000000	1.35914	0.59335
3	0.000000	0.000000	0.45305	0.14030
4	0.000000	0.000000	0.11326	0.02704
5	0.000000	0.000000	0.02265	0.00439
6	0.000000	0.000000	0.00378	0.00062

Table C3: *Estimates of $|\langle\psi, e_n\rangle|^2$ for the Normal and Log-Normal distributions.*

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE	3. SECURITY CLASSIFICATION
Simulation of Statistical Distributions using the Memoryless Nonlinear Transform	Document (U) Title (U) Abstract (U)
4. AUTHORS	5. CORPORATE AUTHOR
Graham V. Weinberg and Lachlan Gunn	Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO–TR–2517	AR-014-934	Technical Report	March 2011

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. No. OF PAGES	12. No. OF REFS
2009/1014219/1	07/040	DGAD	72	25

13. URL OF ELECTRONIC VERSION	14. RELEASE AUTHORITY
`http://www.dsto.defence.gov.au/publications/scientific.php`	Chief, Electronic Warfare and Radar Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DSTO RESEARCH LIBRARY THESAURUS

Clutter Modelling
Correlations
Simulation
Range Profile Generation
Memoryless Nonlinear Transform

19. ABSTRACT

In support of Task 07/040 (support to AIR 7000), the generation of correlated sea clutter returns using the Memoryless Nonlinear Transform is investigated. The generation of such clutter is critical to the performance analysis of radar detection schemes under realistic clutter scenarios. This feature must be incorporated into clutter models being built at DSTO to test radar detector performance. Examples of the transform's application is given for a number of target distributions of interest.